MIPS TECHNOLOGIES

*At the core of the user experience.®*

# Microthreads as Linux CPUs: SMTC Linux for MIPS MT Cores

April 11, 2006

Kevin D. Kissell

MIPS Technologies Proprietary and Confidential

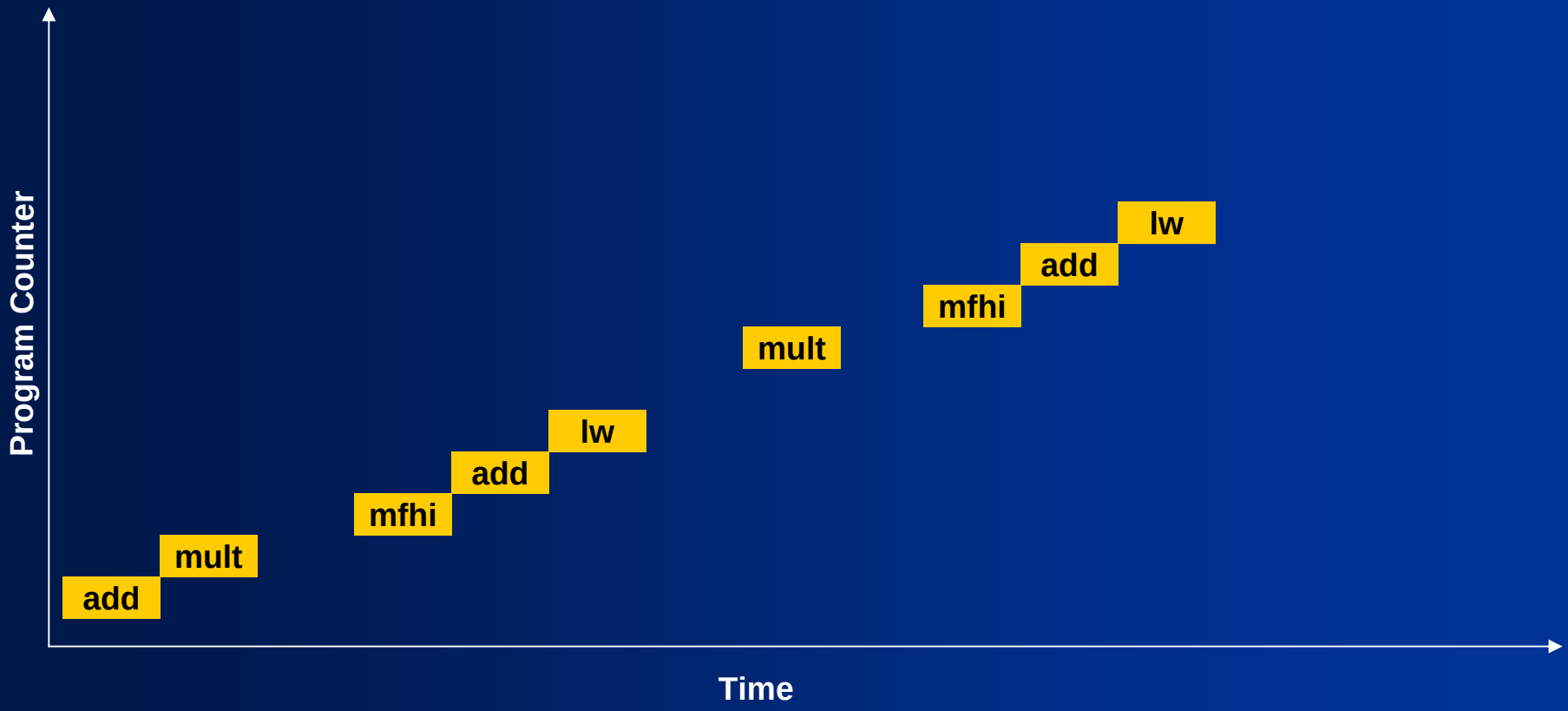
# Agenda

1. Basic Multithreading Concepts
2. The MIPS MT ASE
3. From SMP Linux to SMTC Linux

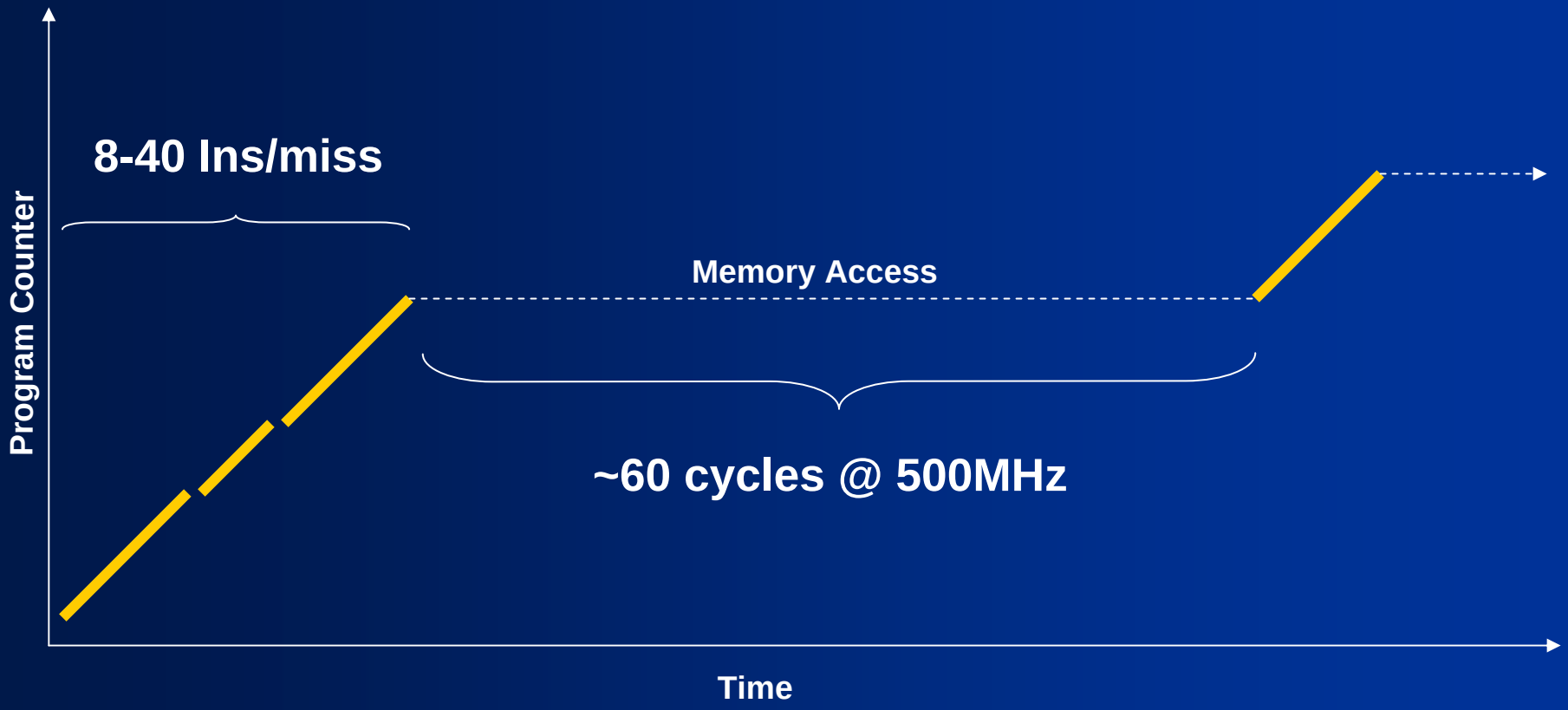
# Limits to "Running Up The Clock"

- Processor Frequency is the Dominant Factor in Performance, but...
- As Processor Frequency Goes Up, Processor Efficiency Goes Down
  - Memory subsystems aren't keeping up – Cache misses are relatively more and more expensive.
  - Key Applications use External Logic and Co-processors with High Access Latencies, e.g. Network Classification Engines
  - ∴ *Increasing Proportion of Processor Bandwidth Wasted on Stalls*
- Best-known Techniques to Improve Efficiency are Expensive in Area and Power for Embedded Applications and Limited in Effectiveness
  - Out-of-order Execution
  - Multilevel Branch and Value Prediction

# How We Visualize RISC Instruction Flow…

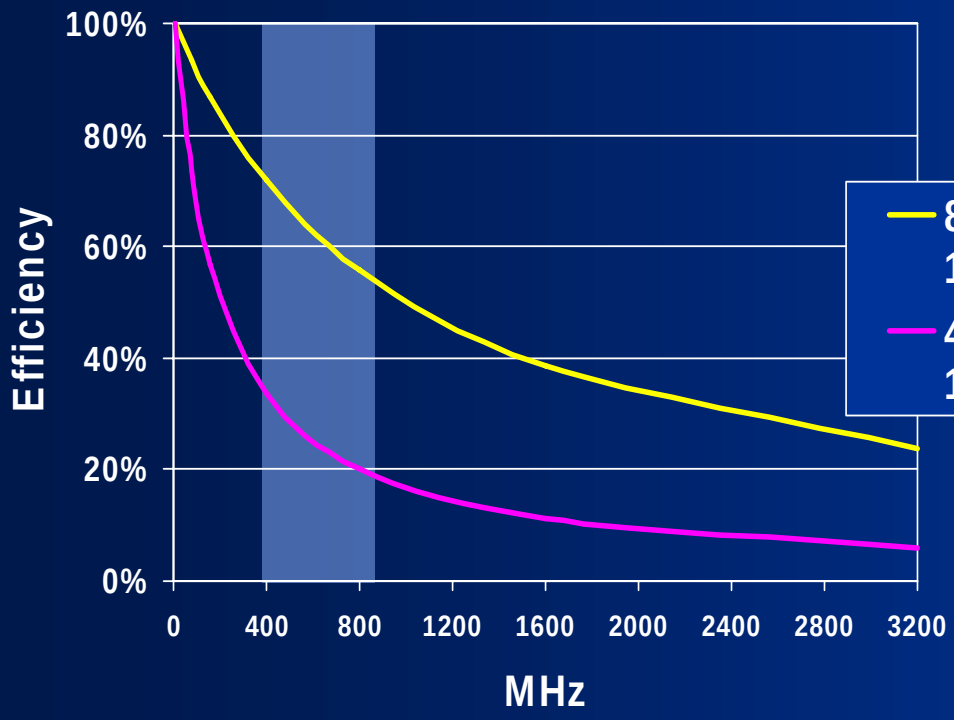


# …And How It Really Behaves.



# Frequency and Memory Latency

➔ Execution speedup with frequency is limited by memory speed

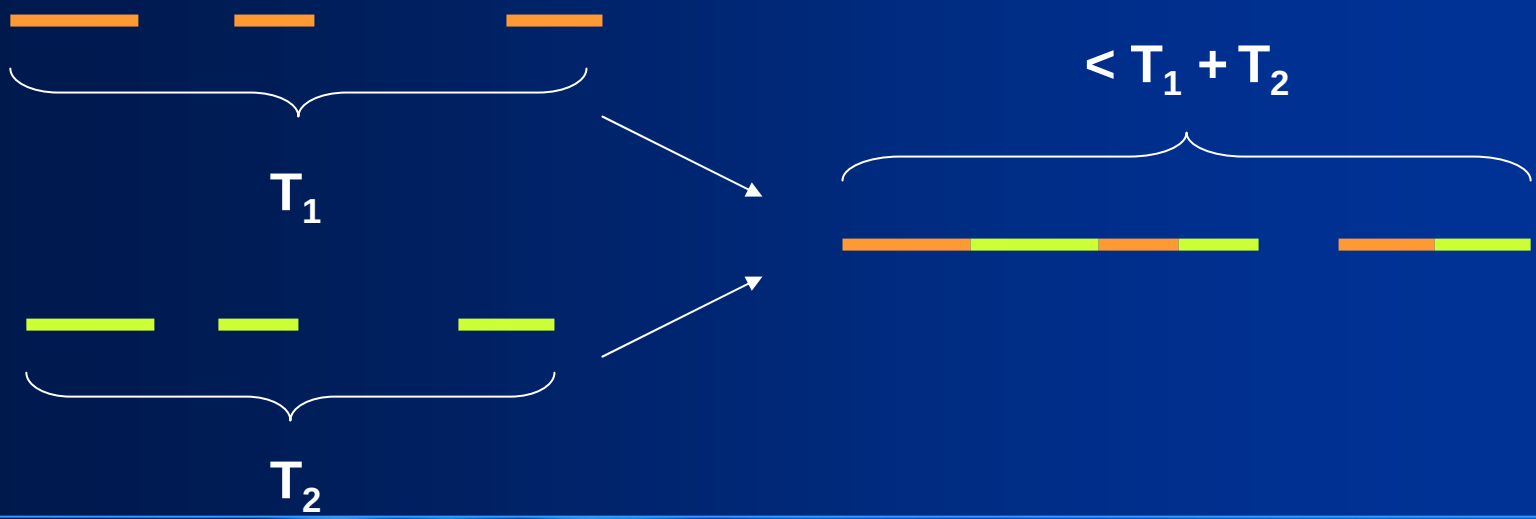



**Assumes 125ns to critical data word**

**"Efficiency" is ratio of "ideal" execution time to expected execution time**

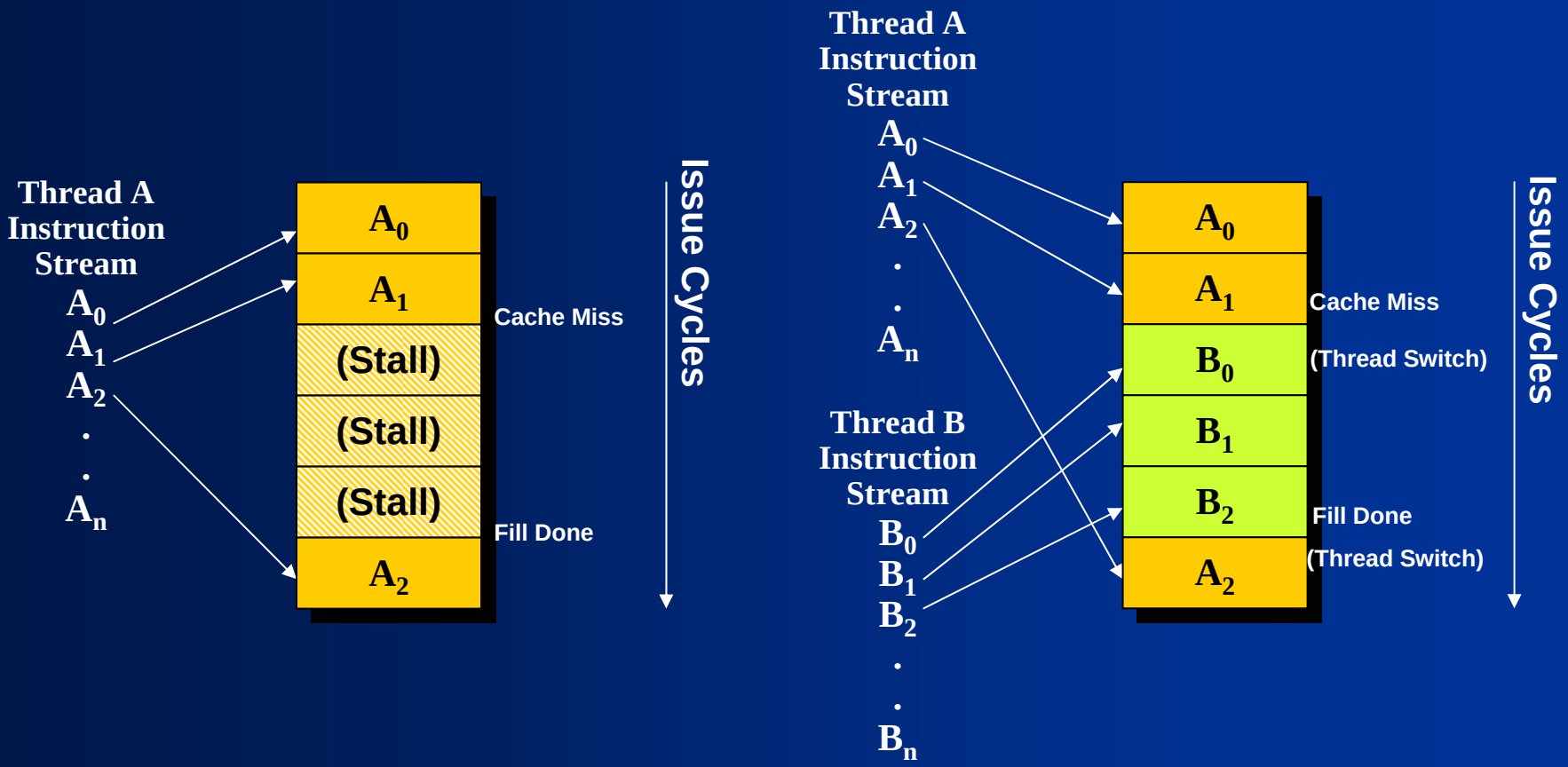
# The Multithreading Hypothesis:

**If Latency Guarantees That A Single Instruction Stream Cannot Keep A Pipeline Busy, Then One Pipeline Can Service More Than One Concurrent Instruction Stream In Less Time Than It Would Take To Run The Instruction Streams Serially**



# Multithreading to Hide Latency

➔ When one instruction stream is blocked, run another



# Multithreading to Reduce Other Overheads

- No need to save previous context, or set up a new one, on an event driven pre-emption
- No need to block all other processing for long periods when interfacing with slow I/O and coprocessors (e.g. Packet Classification Engines)
- Memory-mapped I/O logic (e.g. FIFOs) can drive scheduling

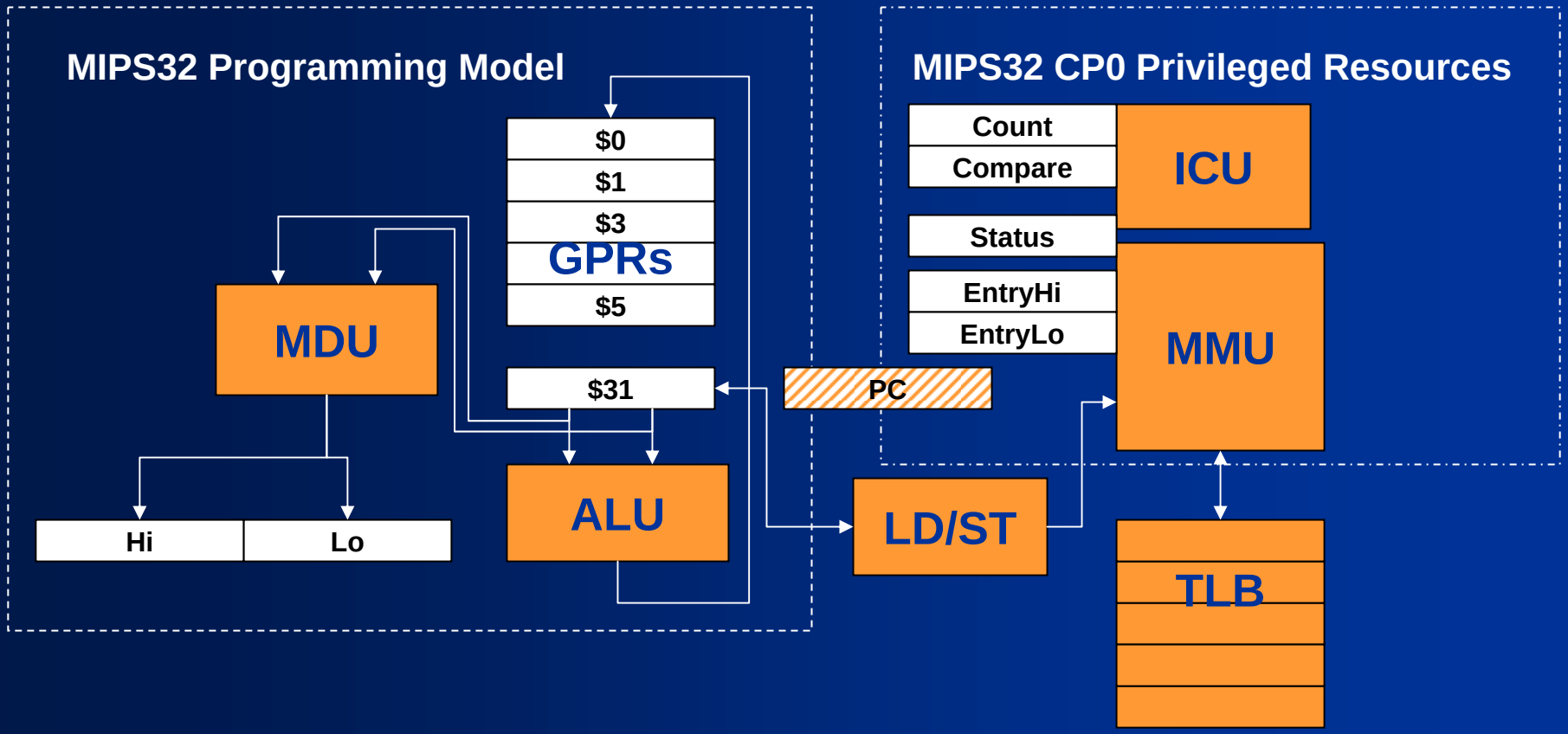
***The "Right" Number of Threads may thus be greater than the number needed to cover for memory latency.***

# The MIPS® MT ASE: Basic Concepts

- A VPE (Virtual Processing Element) is an instance of the MIPS32® or MIPS64® instruction set and privileged architectures. Can run an OS.
- A TC (Thread Context) is an instance of the MIPS32 or MIPS64 application programming model - registers plus access to an address space. Can run a user-mode binary.
- A multi-VPE processor behaves like a closely-coupled SMP configuration, and can run existing SMP OS software.
- A multi-TC VPE supports finer grain parallelism, but requires new OS support for TC management and MIPS MT exceptions.

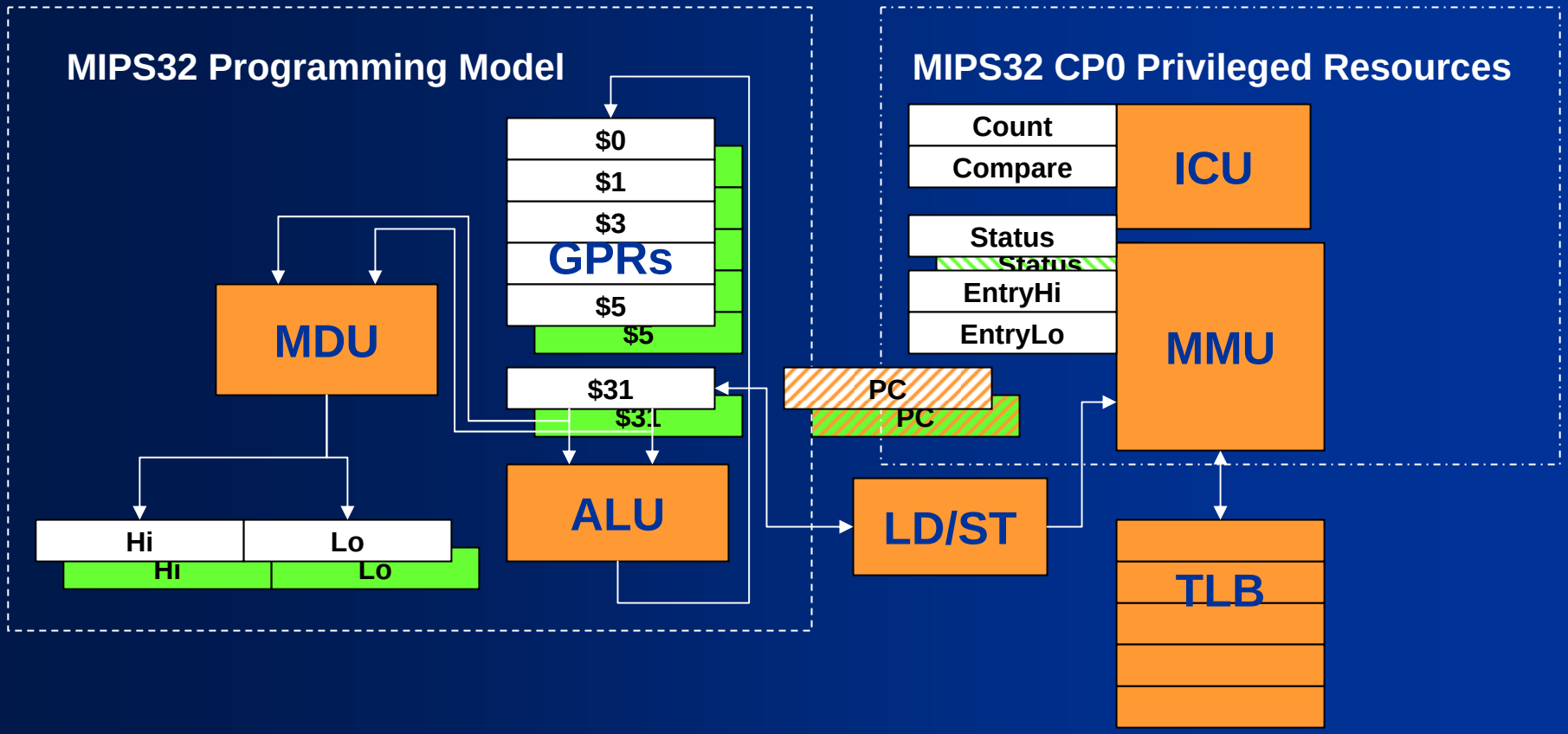
# MIPS32 ISA and Privileged Resources

- Software Visible Processor State Consists of User and PRA Registers



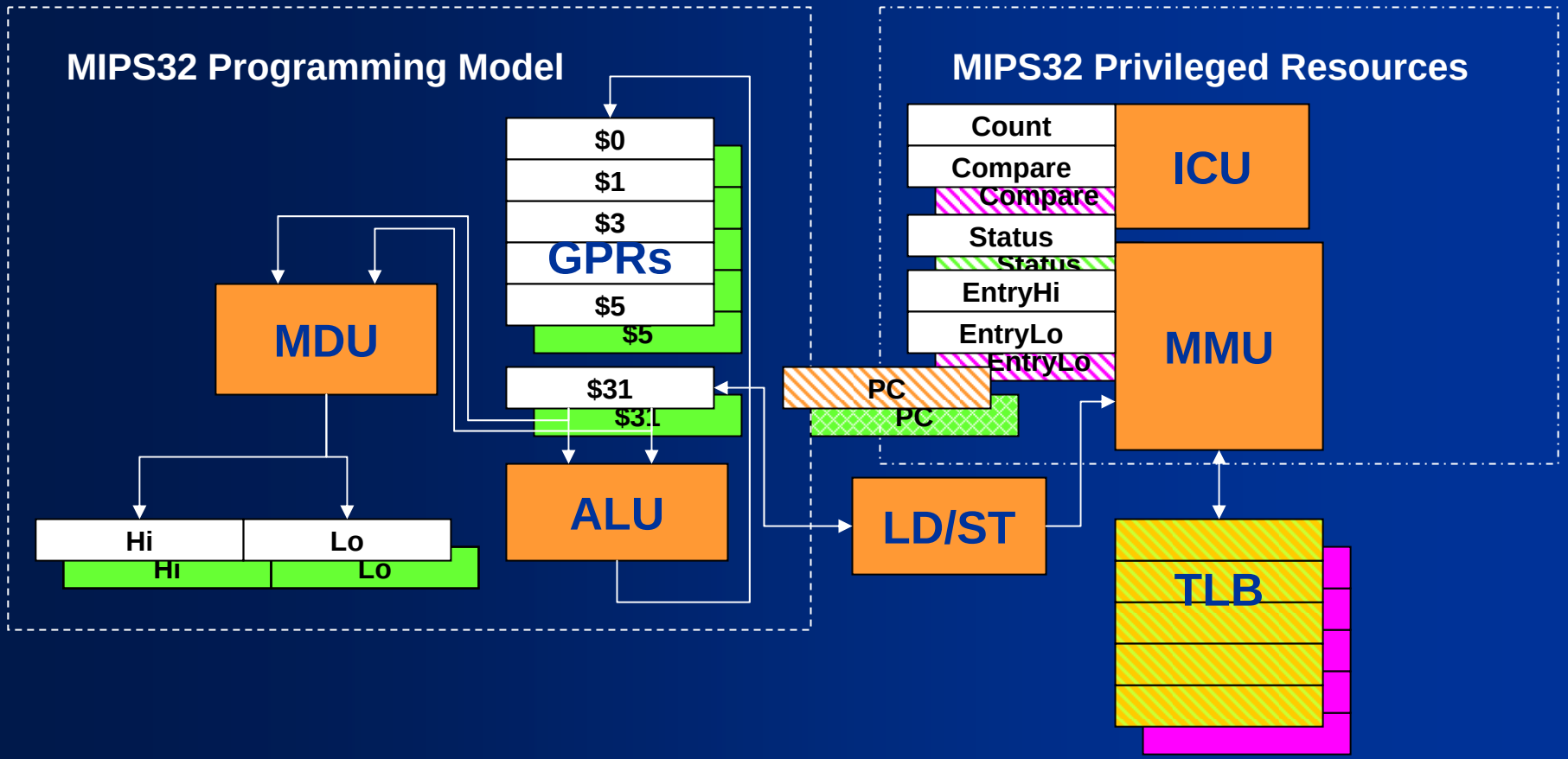
# A Multithreaded Processor with 2 TCs

- Replicate Only What Is Needed for User Mode Concurrency



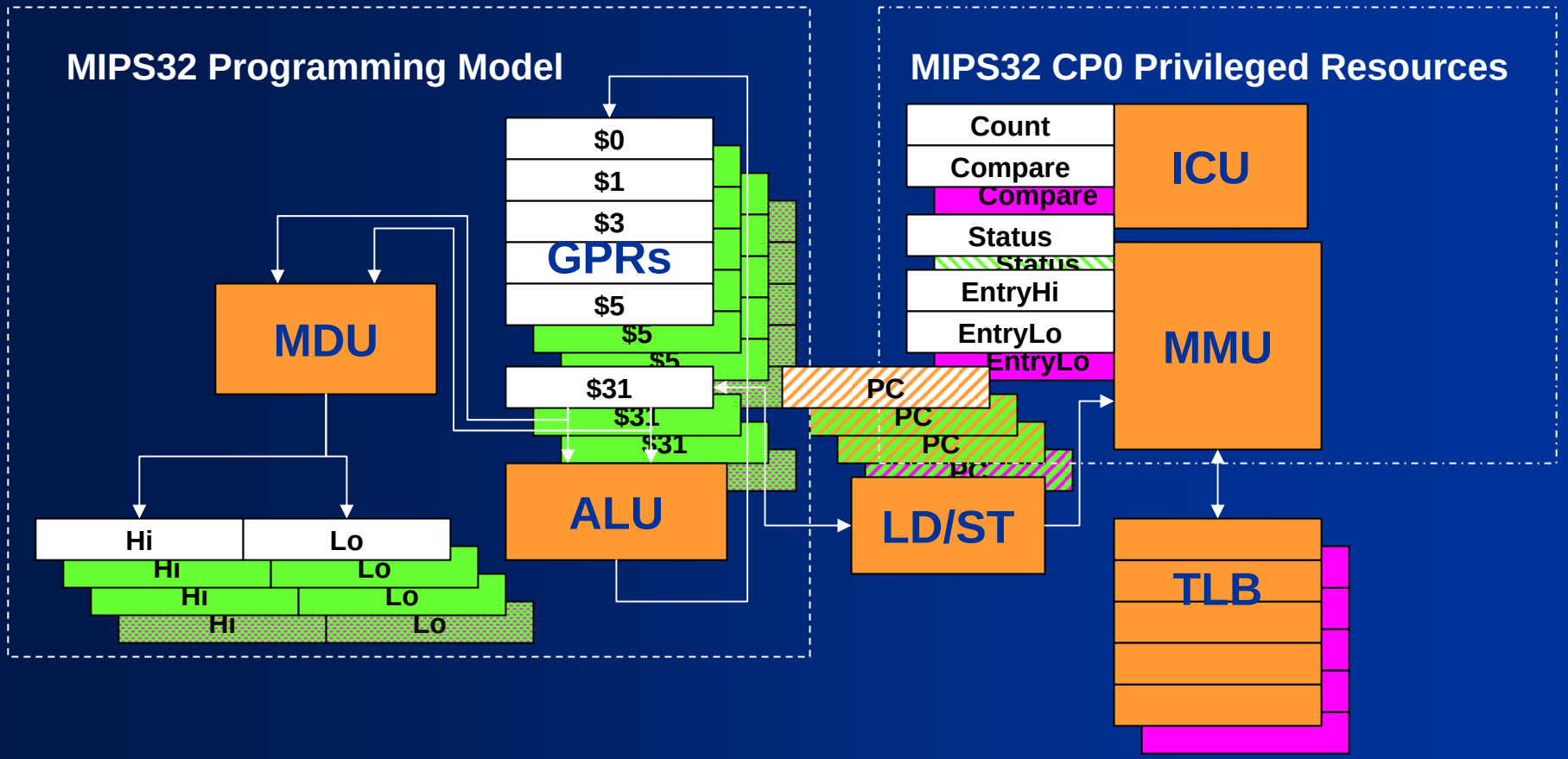
# A Virtual Multiprocessor with 2 VPEs

- Replicate Full "Virtual Processor" State, Sharing Execution Units



# A Multithreaded, Virtual Multiprocessor

- 2 VPEs with 4 TCs Between Them



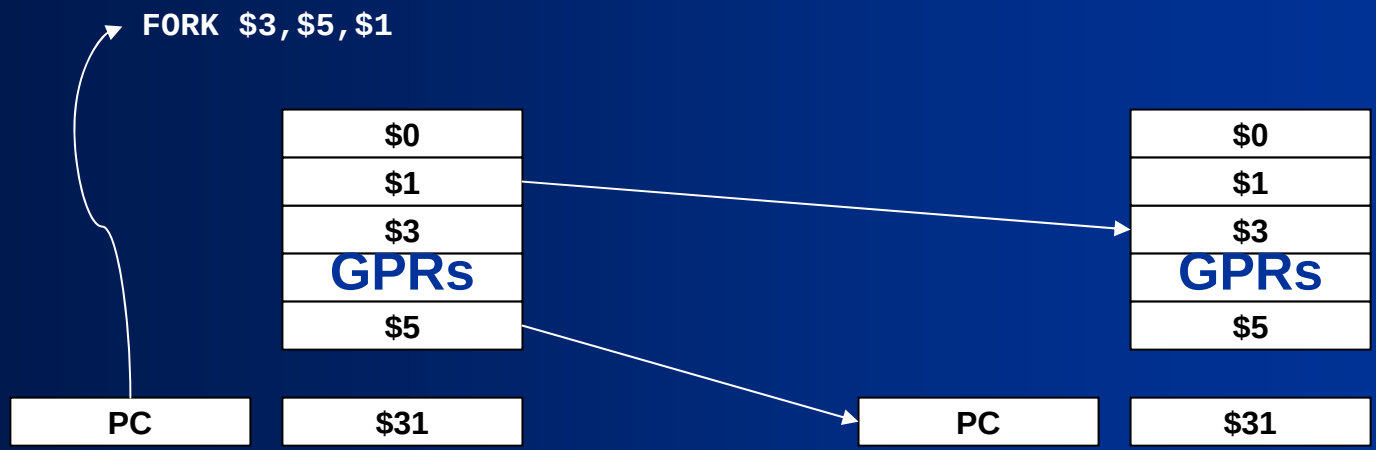
# The MIPS® MT ASE: New Instructions

- FORK allocates a TC and sets it running
- YIELD causes rescheduling/reallocation
- MFTR moves values from a targeted TC to a GPR
- MTTR moves values to a targeted TC from a GPR
- DMT disables multithreading on a VPE
- EMT enables multithreading on a VPE
- DVPE disables multithreading across all VPEs on a processor
- EVPE enables multithreading across all VPEs on a processor

# The MIPS® MT ASE: FORK

FORK rd,rs,rt

Allocate a new TC, to begin execution starting with the instruction at the address in rs with the value in the FORKing TC's register rt copied into the new TC's register rd.



# The MIPS® MT ASE: YIELD

YIELD $0

Terminate thread, de-allocate TC

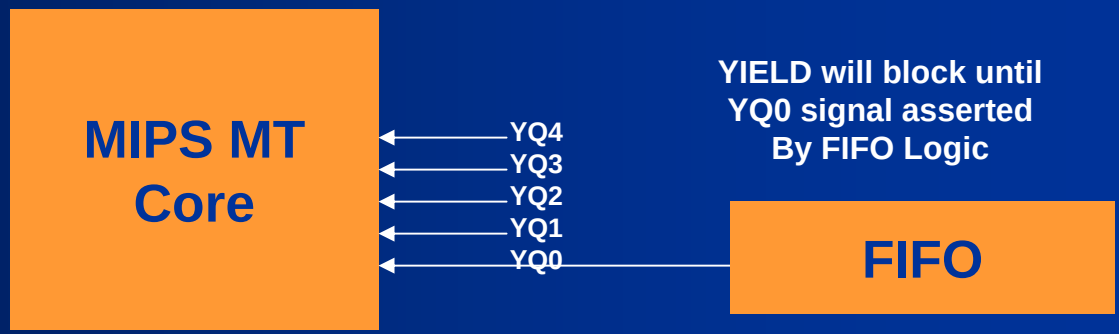
YIELD rd,rs

If rs = -1, update TC schedule and re-run when possible

If rs > 0, re-run when selected YIELD Qualifier inputs are active

rd gets sampled value of YIELD Qualifier inputs

```
# Wait for FIFO full
ADDIU $4,$0,1
YIELD $0,$4
# Now read FIFO
LW $7,0($12)
…
```



# The MIPS® MT ASE: MFTR/MTTR

- Move value to/from another VPE or TC
- TC selected by TargTC field of VPEControl CP0 Register
- Allows access to GPRs, Hi/Lo, CP0, CP1, CP2 registers
- VPE selected is VPE “containing” the target TC
- Cross-VPE accesses allowed only for “privileged” or “master” VPEs

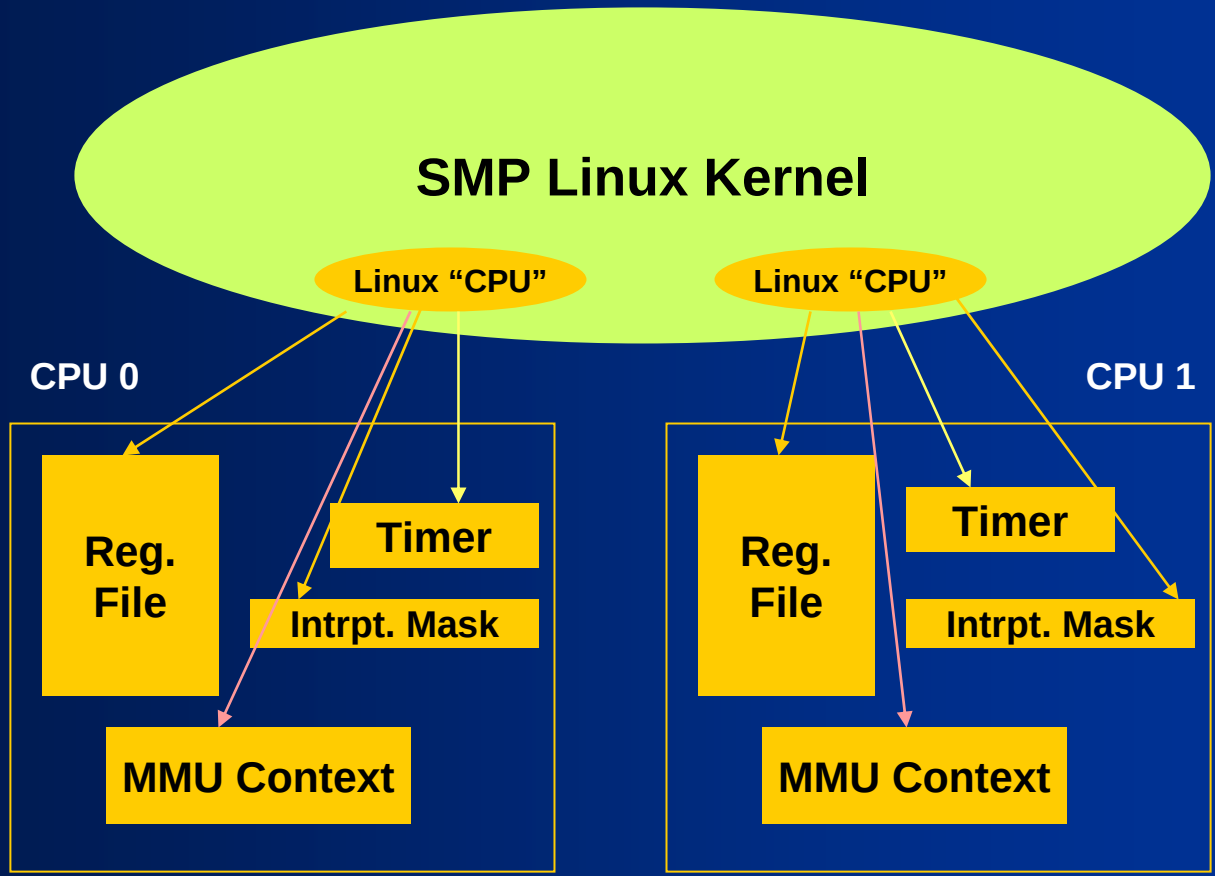
# The MIPS® MT ASE: EMT/DMT, EVPE/DVPE

- Enable/Disable pairs to force serial execution on a single VPE (EMT/DMT) or across all VPEs on a processor (EVPE/DVPE)
- Handy for OS critical regions around use of shared resources
  - Global OS Memory Variables
  - Shared CP0 Resources (e.g. VPEControl)
  - Safe and Easy to Use within Exception Handlers
- EVPE/DVPE allowed only for “privileged” or “master” VPEs

# The MIPS® MT ASE: Exceptions

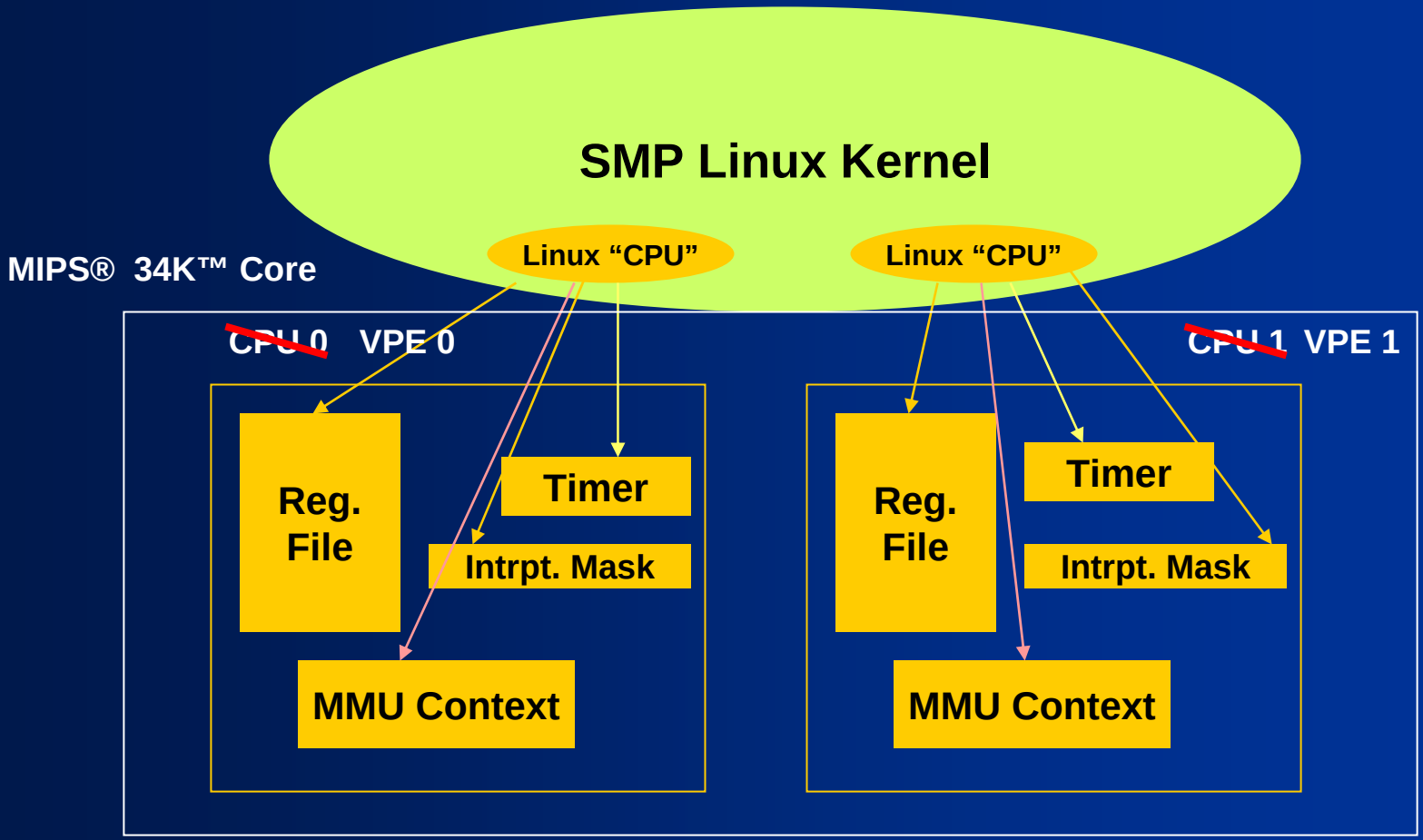
- Exception Logic Instantiated per-VPE
  - VPEs can service exceptions concurrently
- Exception Logic Shared among TCs on a VPE
  - Setting EXL/ERL or entering Debug mode suspends all TCs except the one servicing the exception
  - ERET re-enables multithreaded operation of a VPE

# Linux Kernel Abstraction of an SMP Processor



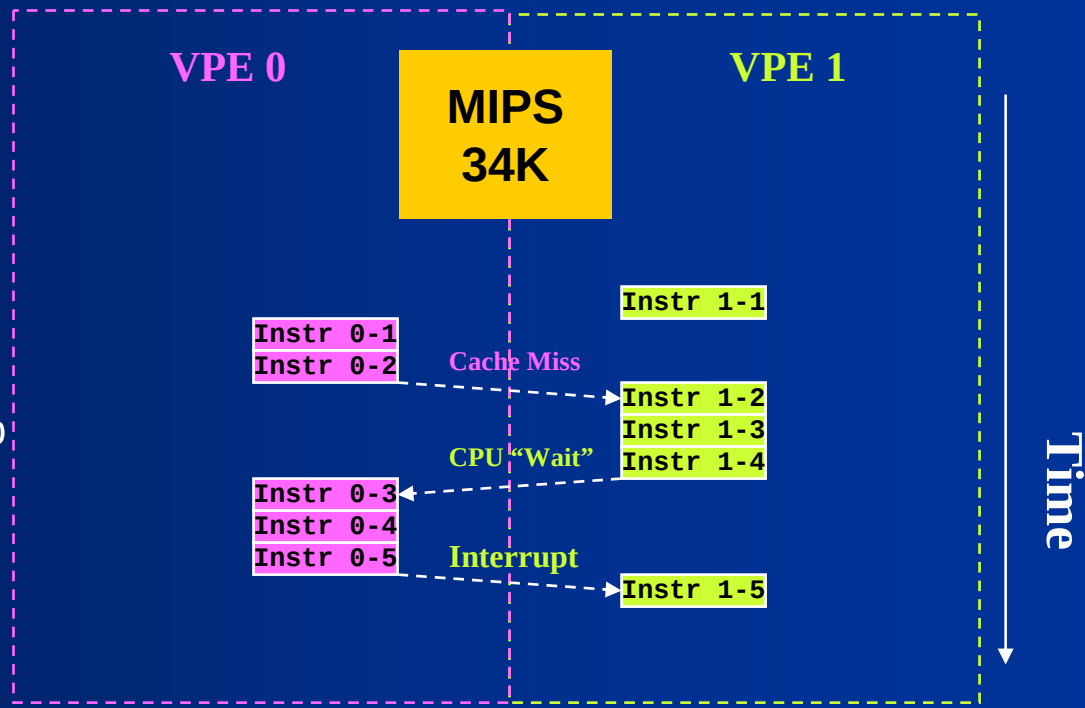

**Hardware State that Defines a “CPU” to software**

# SMP Linux Kernel on MIPS MT VPEs (SMVP)

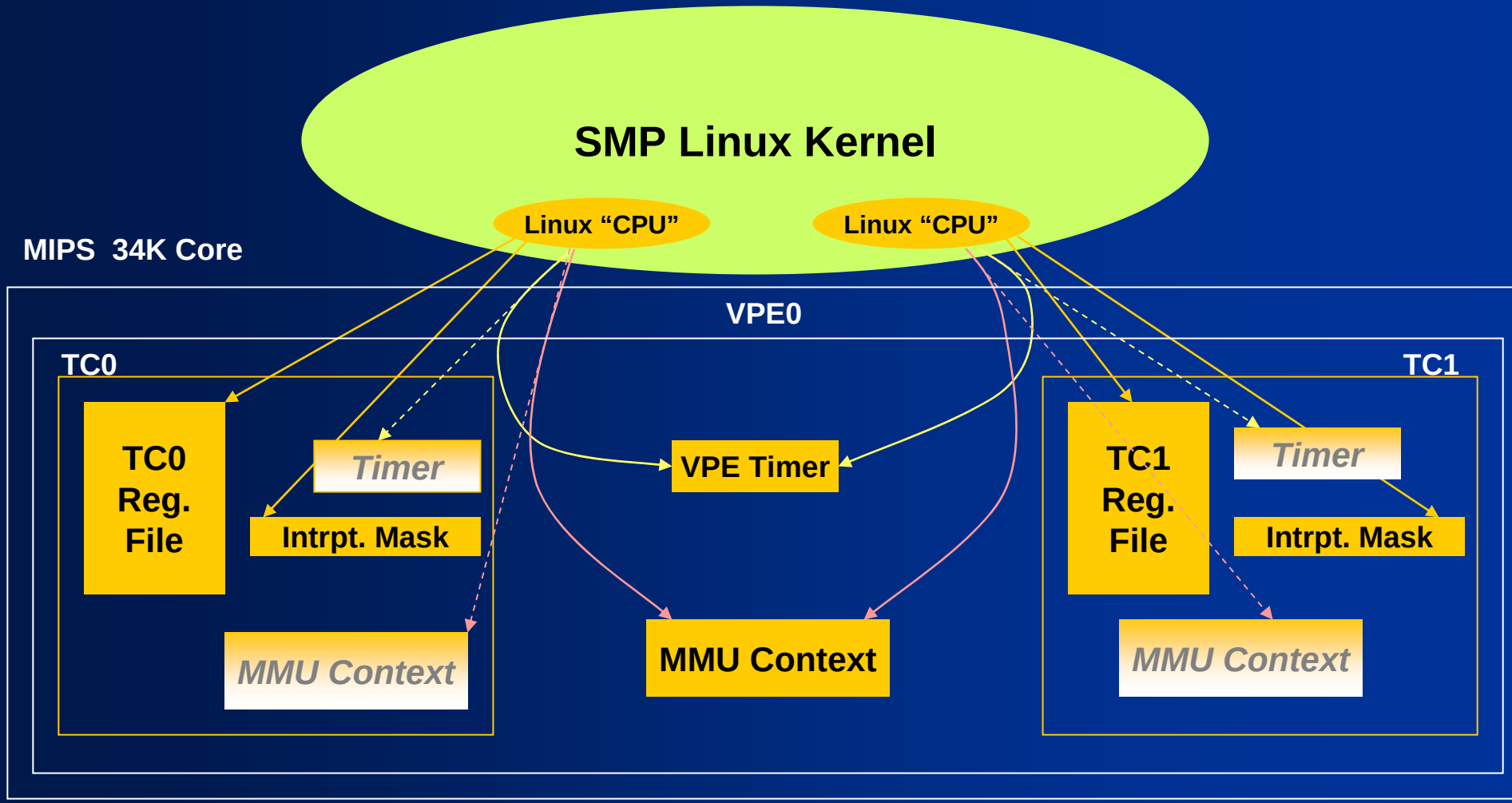


# Linux for Symmetric Multiple Virtual Processors:
## Multitasking Throughput with Legacy Binaries

- Each program/thread sees a different MIPS32 “Processor”
- SMP Linux Schedules tasks on multiple VPEs as if they were CPUs
- Speedup less than SMP, but significant
- Less performance gain but much lower area, power consumption, than multi-core
- For MIPS® 34K™, performance gains far outweighs costs (e.g. 60% speedup for 14% area growth)
- But limited to 2-way SMP

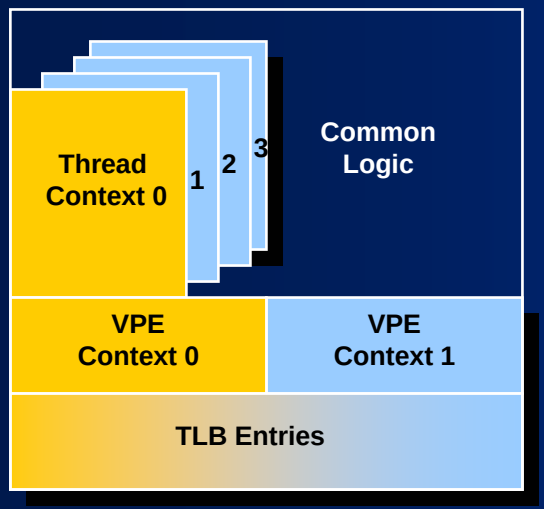


# SMTC on one VPE



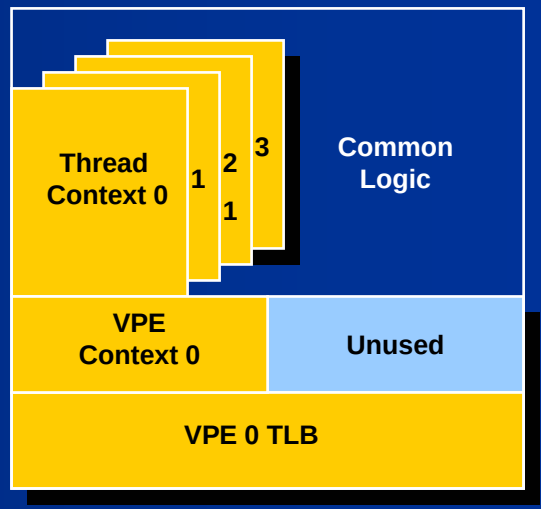
# Optimizing MIPS® MT VPE/TC Configuration

- 34K Core TCs Assigned to VPEs at Boot Time

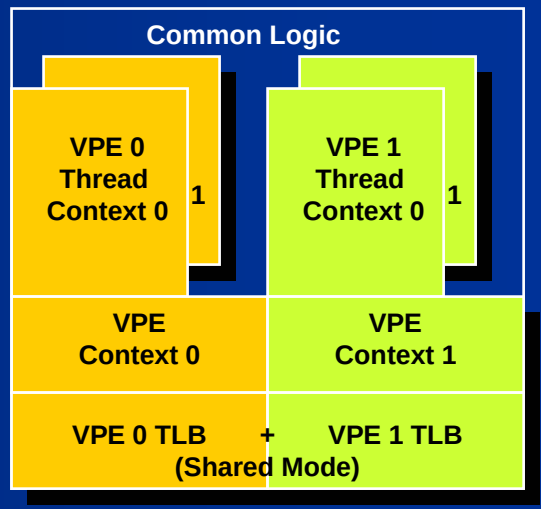



**Reset State**

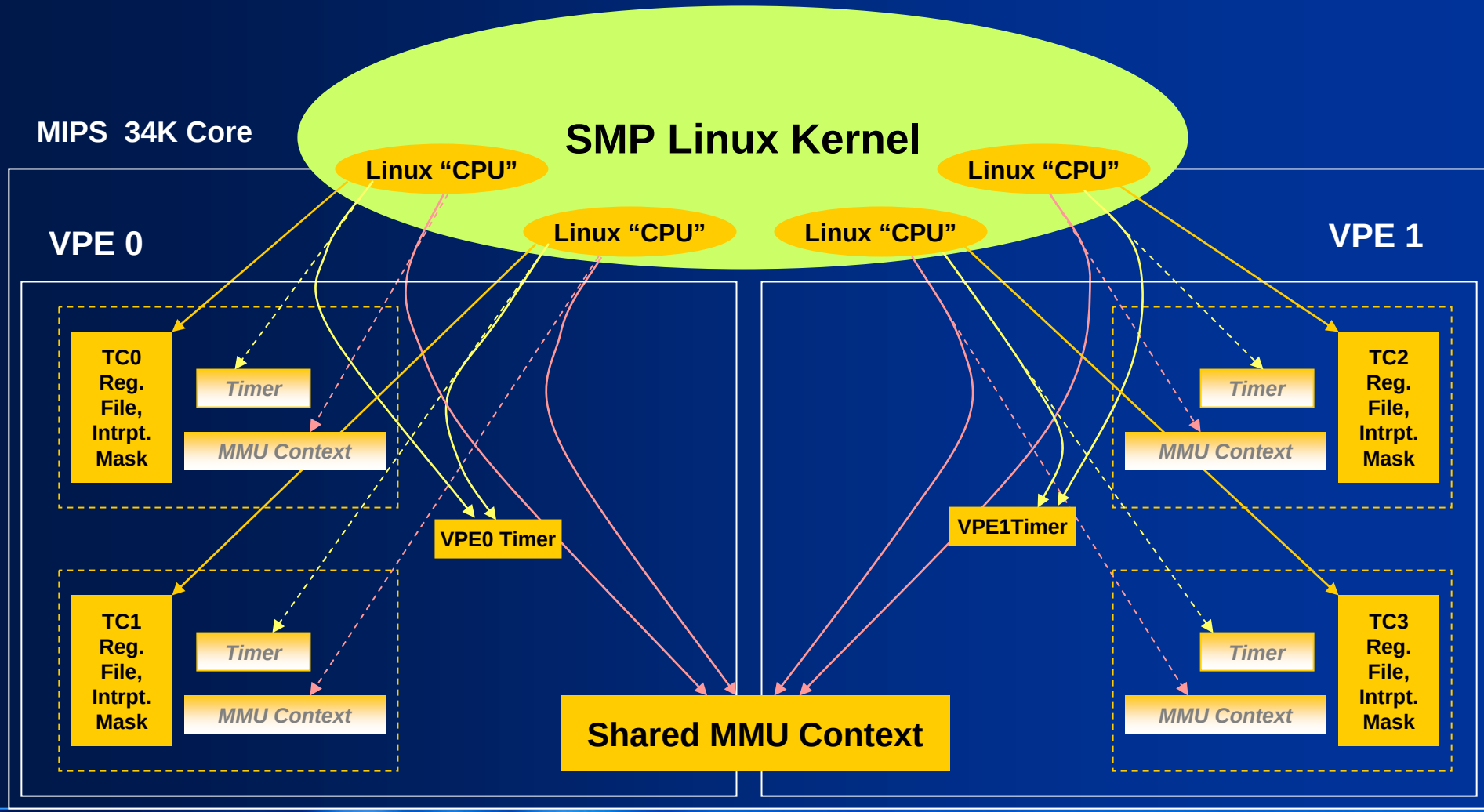
Bind 4 Threads and all TLB Entries to VPE 0




Bind 2 Thread and half of TLB Entries to each of VPE 0 and VPE 1



# SMTC on Multiple VPEs



# SMTC: The Easy Parts

- Set up a cpu_data array entry for each TC, add new tc_id and vpe_id fields.
- In kernel macros, replace use of standard MIPS Status.IE bit for interrupt enable/disable with manipulation of per-TC TCStatus.IXMT
- Have SMP initialization set up each “secondary” TC to start up executing smp_bootstrap code.
- Protect read-modify-writes of per-VPE registers (e.g. Status) and cache/TLB management operations with lock or DMT-based critical regions.

# SMTC Challenges: Interrupt Management

- **"Convoying" of Threads into Interrupt Handlers**
  - Interrupt inputs to MIPS MT core are per-VPE, serviced opportunistically by available TCs
  - Standard MIPS Linux kernel passes from "EXL" exception state to local CPU interrupt disable state before any "ack" by handler. Clearing VPE-wide exception state allows other TCs to run and attempt service
- **Solution: Selectively Inhibit Interrupt Levels per-VPE during IRQ service dispatch**
  - While still in exception state, clear the VPE-level Interrupt Mask bit for the interrupt under service
  - Optimally, restore mask in SMTC-aware interrupt controller "ack(irq,…)" function
  - If not handled explicitly, mask is restored on return from interrupt

# SMTC Challenges: Inter-Processor Interrupts

- **SMP Linux kernel requires that IPIs be directed to any/all "CPUs"**
  - Software interrupts, like HW inputs in MIPS MT, are per-VPE
- **Solution: Use MIPS MT MFTR/MTTR instructions to emulate per-TC interrupts**
  - IPI dispatch between TCs of a VPE done by halting target, extracting, pre-saving, and modifying some state to force kernel mode execution of pseudo-vector, then un-halting.
  - More work by sender than conventional I/O operations to activate a physical interprocessor interrupt, but less work by receiver, as interrupt is "pre-decoded" to an IPI vector.
  - If TC is in an interrupt-inhibited state, IPI is queued and picked up on next interrupt, context restore, or entry into kernel idle loop.

# SMTC Challenges: Shared TLB, ASIDs

- **SMP Linux kernel sensibly assumes each CPU has its own TLB**
  - Address space Ids (ASIDs) must be unique across a VPE to avoid conflicts in shared TLB
  - Shared ASID space means more than one value may be “live” at one time, breaking assumptions in allocation/generation code.
- **Solution: Explicitly Manage ASIDs as a pooled resource**
  - Replace use of per-CPU ASID on each memory map with a single ASID across all “CPUs”.
  - On “roll-over” of ASID “generation”, generate table of “surviving” ASIDs that are still in use for other TCs, and keep them from being reused until they are retired from the TLB previous use.
  - Reduction in TLB “thrashing” by having a single entry usable by all TCs offsets overhead of global management per-VPE.

# SMTC Challenges: FPU Context Management

- **MIPS 34Kf has Single FPU Context but up to 5 TCs**
  - Only one TC can be allowed to issue FP instructions at a time
  - Difference between FPU execution and kernel FP emulation is a factor of roughly 100 in performance
- **Solution: Heuristic-based dynamic FPU Affinity**
  - Exploits CPU affinity hooks already in SMP Linux scheduler
  - When more than N FPU emulations are done by a Linux thread in a single time-slice, make it eligible to run only on TC with FPU
  - When no FP instructions have been issued during a time-slice by a thread with such FPU affinity, remove constraint.
  - Provides near-optimal FP performance of legacy binaries with very low overhead.

# Summary

- SMTC Linux for MIPS MT builds on robust existing MIPS SMP Linux support
- SMTC supports a larger number of virtual processors at a lower hardware cost per processor
- Overhead of SMTC support in Linux is measurable but far smaller than the potential performance gain of the greater concurrency
- SMTC imposes no application code changes to support thread libraries (NPTL, pthreads) on top of hardware TCs
- SMTC Linux kernel sources for the 34K will be available soon at www.linux-mips.org (if they aren't already).